Course #2:

Deep Learning, from MLP to CNN

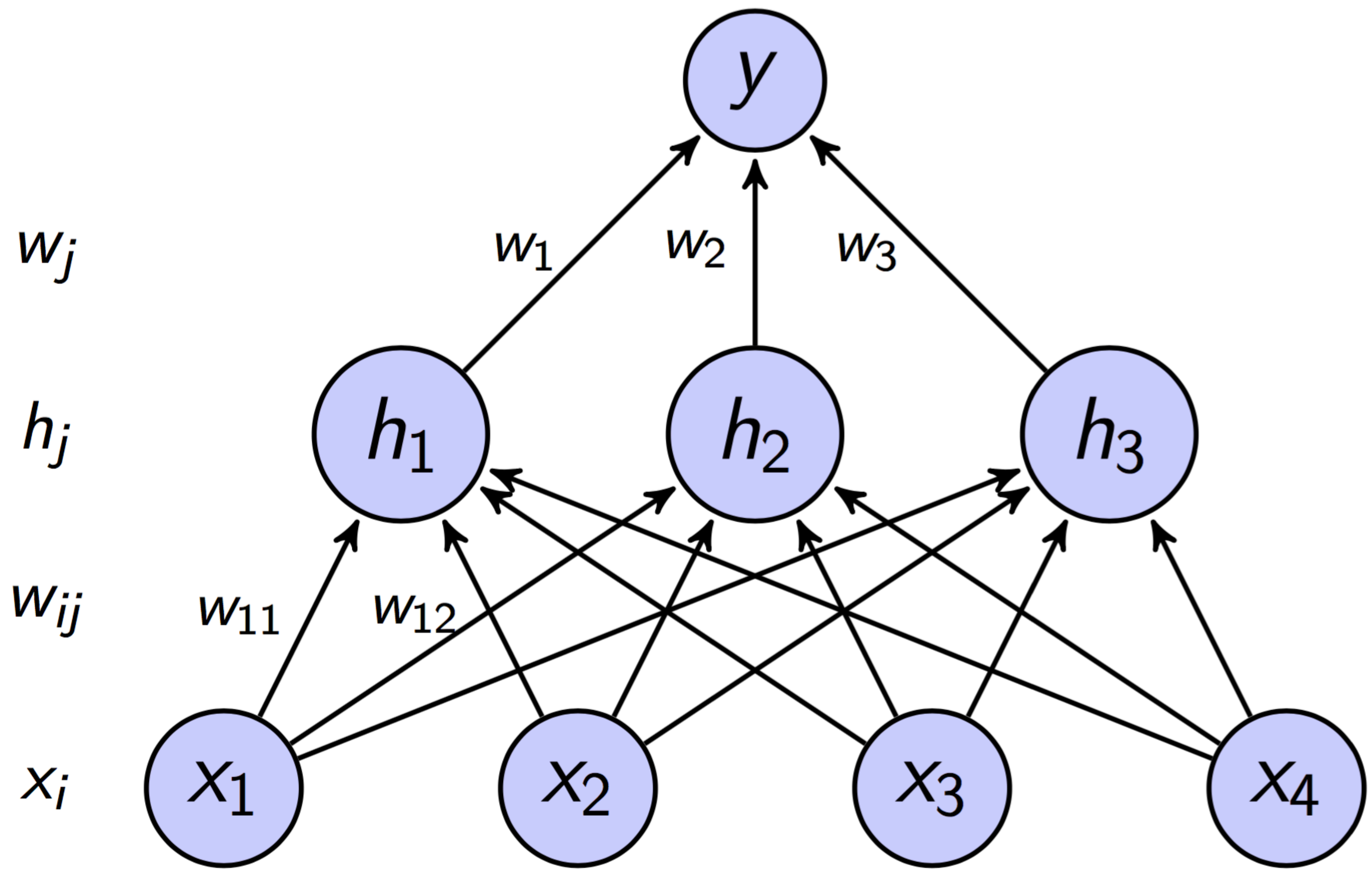
Roadmap

- Recap from course #1
- MLP and Image classification as a case study
- CNN: basic principles
- Application to image classification
- Classic CNN architectures

Recap from Course #1
Things to know

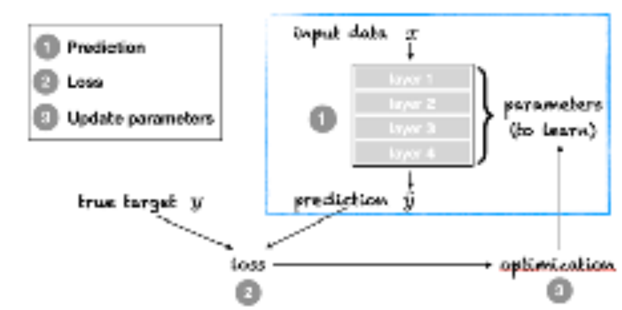
- Supervised vs. unsupervised learning
- Training loss
- Model
- Layers
- Fully-Connected/Dense NNs (MLP)
- Activation functions
- Backpropagation
- Weights and biases
- Optimizers
- epoch

Feedforward networks
(Weights and biases)



$$f(x) = \sigma\left[\sum_i \omega_i \sigma_i \left(\sum_j \omega_{i,j} x_j + b_i\right) + b\right]$$

Guidelines to implement Deep Learning schemes



1. Problem formulation (inputs/outputs)
2. Data collection (cf. supervised vs. non-supervised)
3. Definition of performance metrics
4. Selection of neural architectures (at least 2 models)
5. Selection of a training loss
6. Split dataset into training / validation / test datasets
7. Train the selected models from the training dataset and save the best models onto the validation dataset
8. Benchmark the performance of the trained models onto the test dataset
9. Update/iterate 4-5-6-7-8

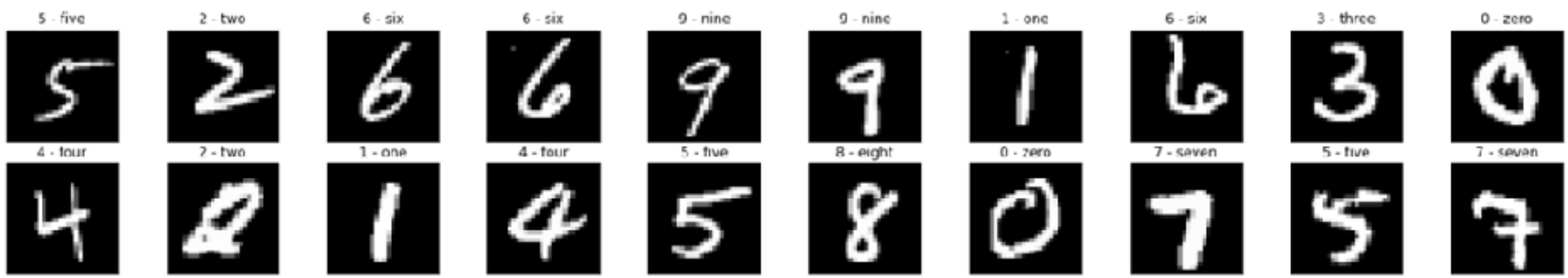
Image classification case-study

Let's go

https://github.com/CIA-Oceanix/DLCourse_MOi_2022/blob/main/notebooks/notebook_MNIST_classification_MLP_with_correction.ipynb

Image classification case-study with pytorch

1. Problem formulation (inputs/outputs)



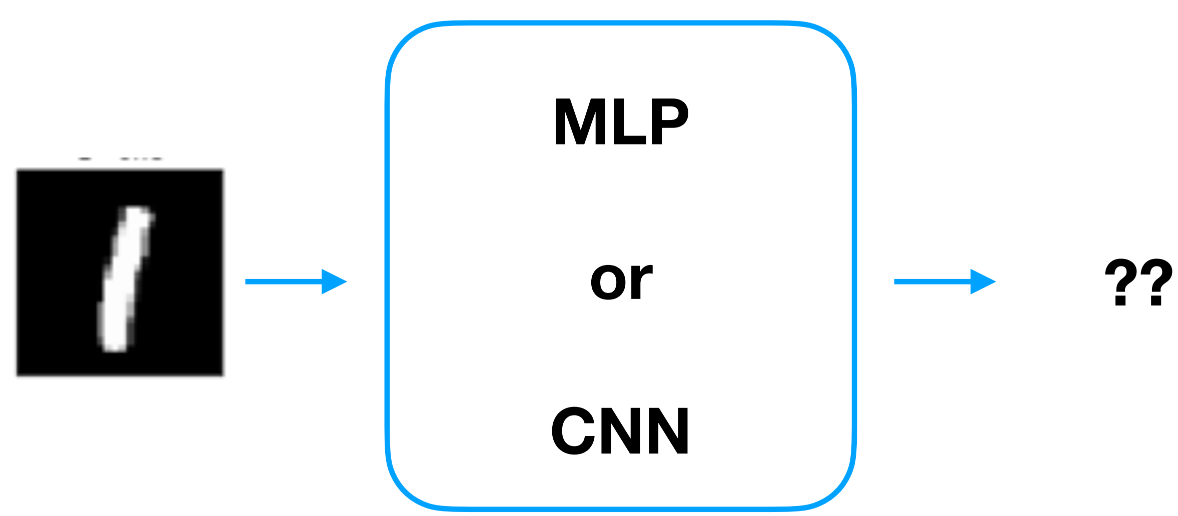


Image classification case-study with pytorch

Training / validation / test dataset

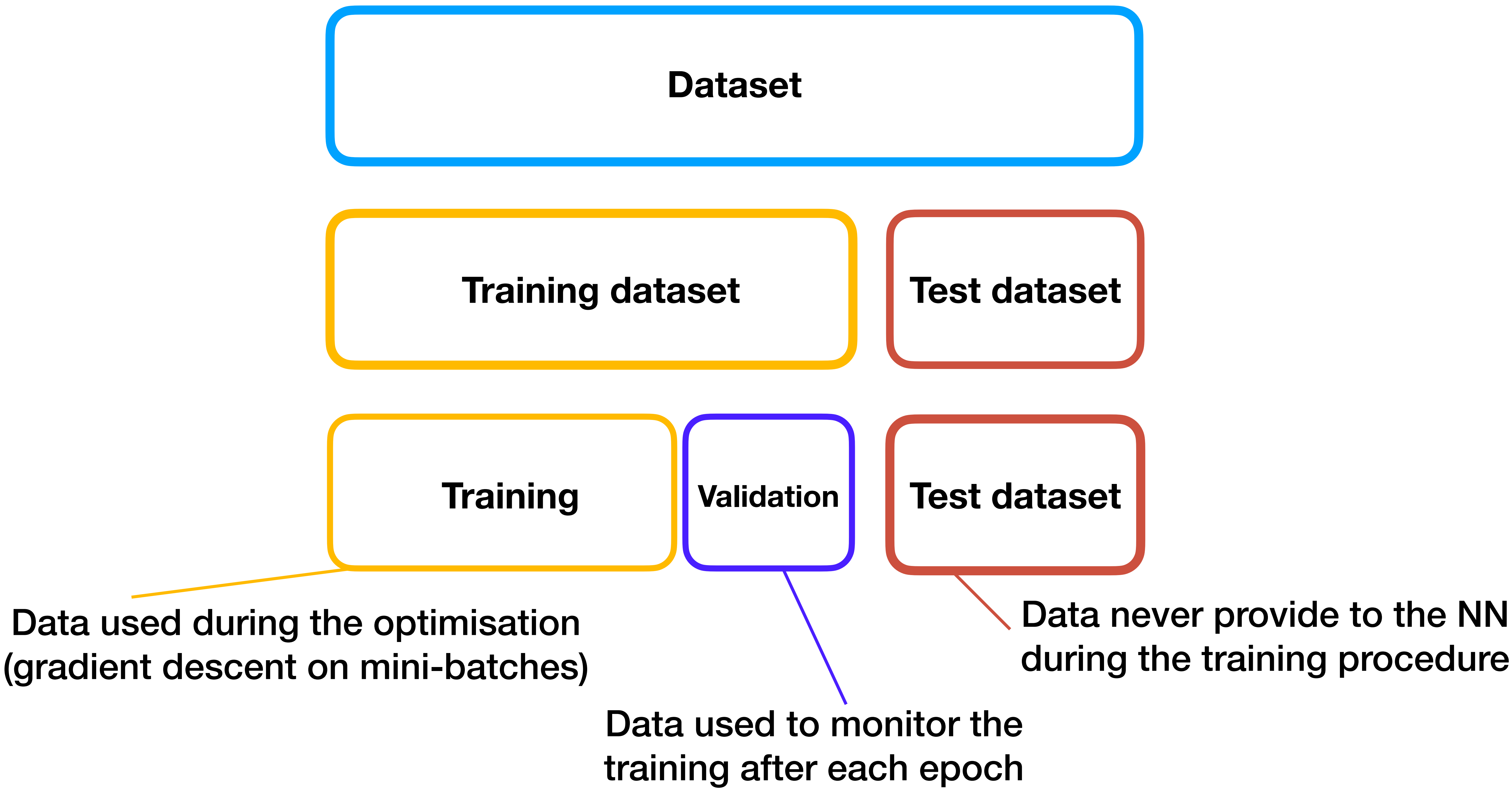


Image classification case-study with pytorch

2. Data collection

```
train_data = datasets.MNIST(root = 'data', train = True, download = True, transform = transform)
test_data = datasets.MNIST(root = 'data', train = False, download = True, transform = transform)
```

3. Performance metrics

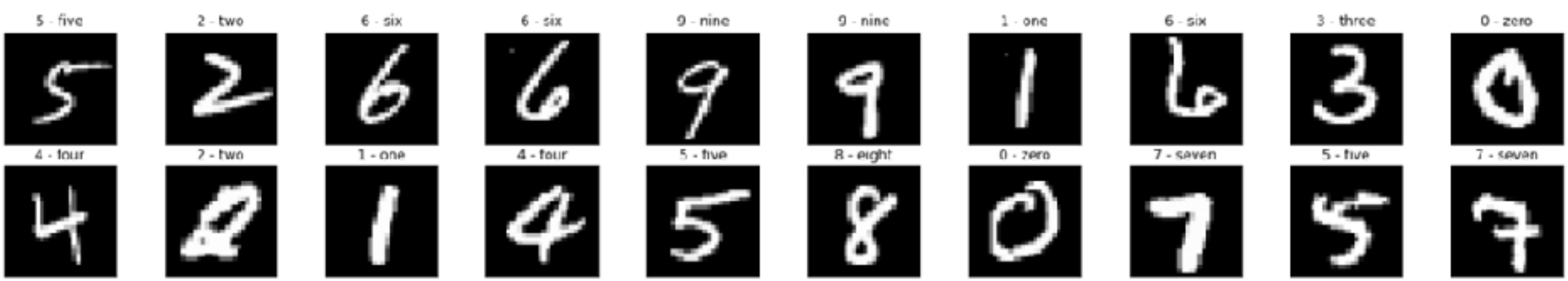


Image classification case-study with pytorch

4. Neural architecture

```
import torch.nn as nn
import torch.nn.functional as F

class MLP(nn.Module):
    def __init__(self): # FUNCTION TO BE COMPLETED
        super(MLP,self).__init__()
        hidden_1, hidden_2 = 512, 256
        self.fc1 = nn.Linear(28*28, hidden_1)
        self.fc2 = nn.Linear(hidden_1,hidden_2)
        self.fc3 = nn.Linear(hidden_2,10)
        self.dropout = nn.Dropout(0.2)

    def forward(self,x): # FUNCTION TO BE COMPLETED
        x = x.view(-1,28*28)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        return x
```

5. Training loss

```
criterion = nn.CrossEntropyLoss() # TO DO
```

Model complexity ?

Image classification case-study with pytorch

6. Split dataset into training / validation / test datasets

```
import torch
from torch.utils.data.sampler import SubsetRandomSampler
import numpy as np

batch_size = 20
valid_size = 0.2

train_size = 0.2
indices = np.random.permutation(len(train_data))[:int(train_size*len(train_data))]
train_data = torch.utils.data.Subset(train_data,indices )

def create_data_loaders(batch_size, valid_size, train_data, test_data): # FUNCTION TO BE COMPLETED

  total_train = len(train_data)
  num_val = int(total_train * valid_size)
  num_train = total_train - num_val

  tr_data, val_data = torch.utils.data.random_split(train_data, [num_train, num_val])
  train_loader = torch.utils.data.DataLoader(tr_data, batch_size = batch_size)
  valid_loader = torch.utils.data.DataLoader(val_data, batch_size = batch_size)
  test_loader = torch.utils.data.DataLoader(test_data, batch_size = batch_size)

  return train_loader, valid_loader, test_loader
```

Image classification case-study with pytorch

7. Model training

```
optimizer = torch.optim.SGD(model_1.parameters(),lr = 0.01)
```

```
for epoch in range(n_epochs):
    train_loss, valid_loss = 0, 0

    model.train()
    for data, label in train_loader:
        data = data.to(device=device, dtype=torch.float32)
        label = label.to(device=device, dtype=torch.long)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, label)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * data.size(0)

    model.eval()
    for data, label in valid_loader:
        data = data.to(device=device, dtype=torch.float32)
        label = label.to(device=device, dtype=torch.long)
        with torch.no_grad():
            output = model(data)
        loss = criterion(output,label)
        valid_loss += loss.item() * data.size(0)

    train_loss /= len(train_loader.sampler)
    valid_loss /= len(valid_loader.sampler)
    train_losses.append(train_loss)
```

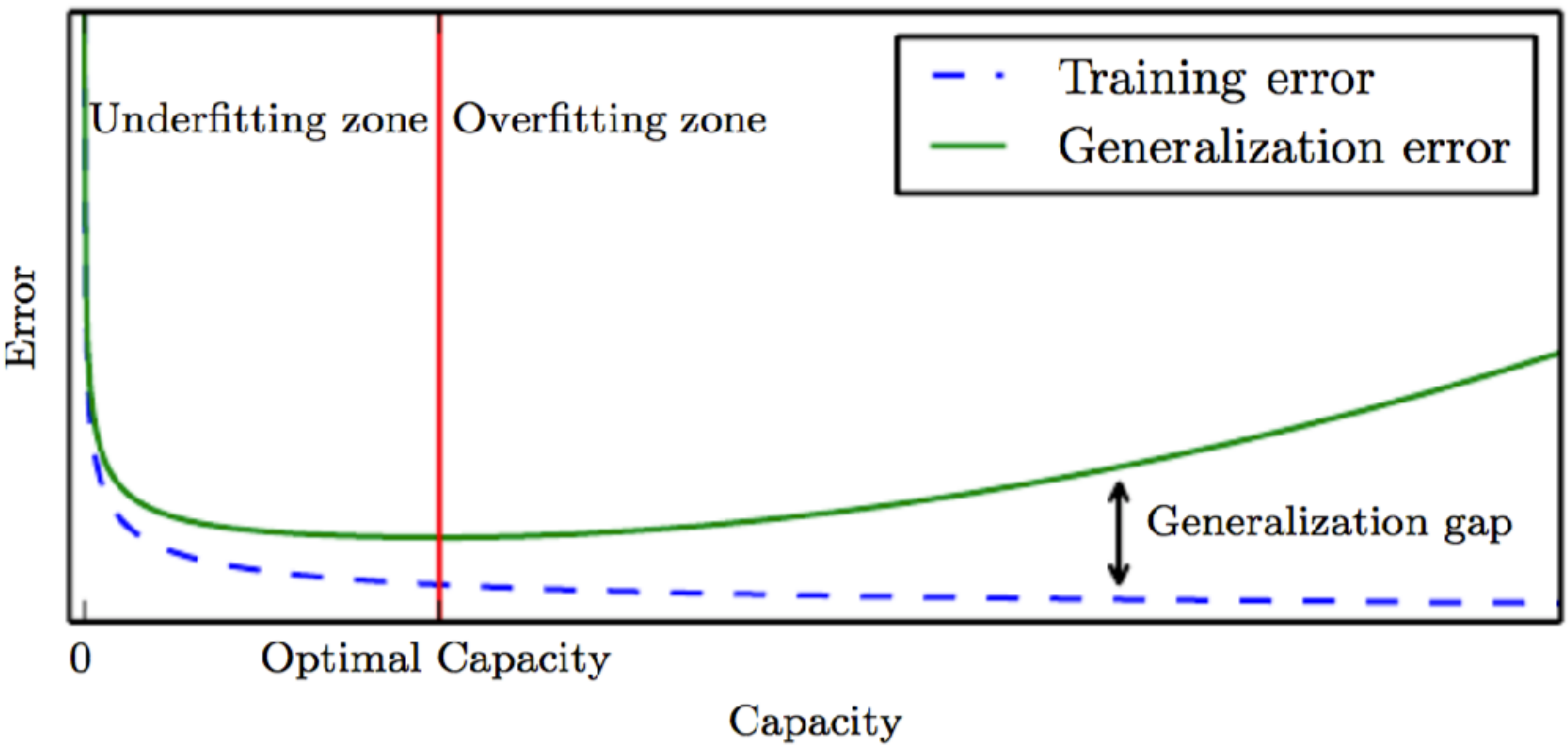
Image classification case-study

Go and run the notebook

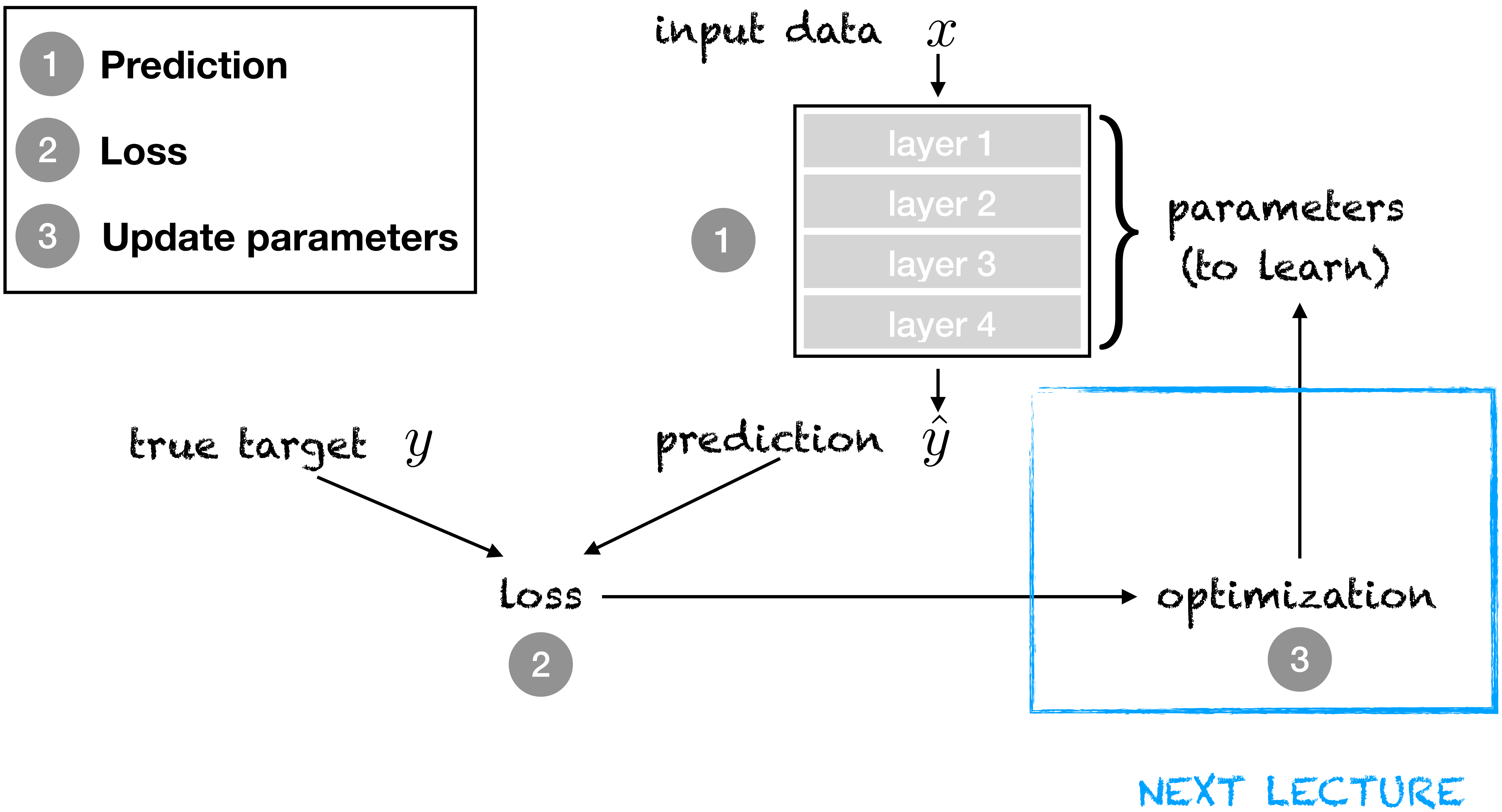
Questions :

Test the training procedure for the MLP with a dropout value of 0. and 0.2.
What is the effect of the dropout layer ?

Over-fitting



Overview



1 Prediction
2 Loss
3 Update parameters
input data x
layer 1
layer 2
layer 3
layer 4
1
parameters (to learn)
true target y
prediction $\hat{y}$
loss
2
optimization
3
NEXT LECTURE

Optimizers

[Chapter 8, Goodfellow et al.]

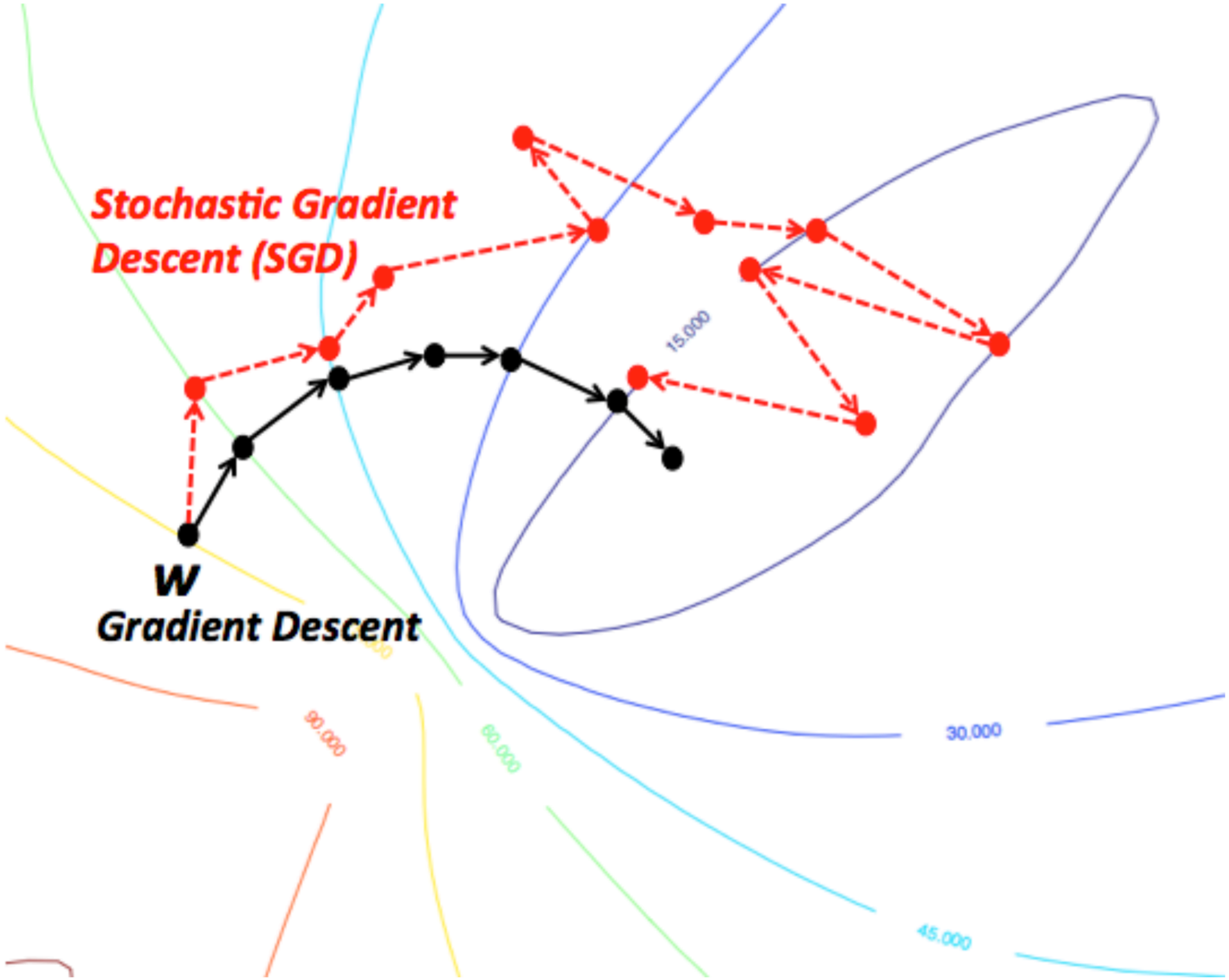
Gradient-based approach

- Stochastic gradient descent (i.i.d examples):

$$\theta^{k+1} = \theta^k - \epsilon_k \frac{\partial J(\theta^k)}{\partial \theta^k}$$

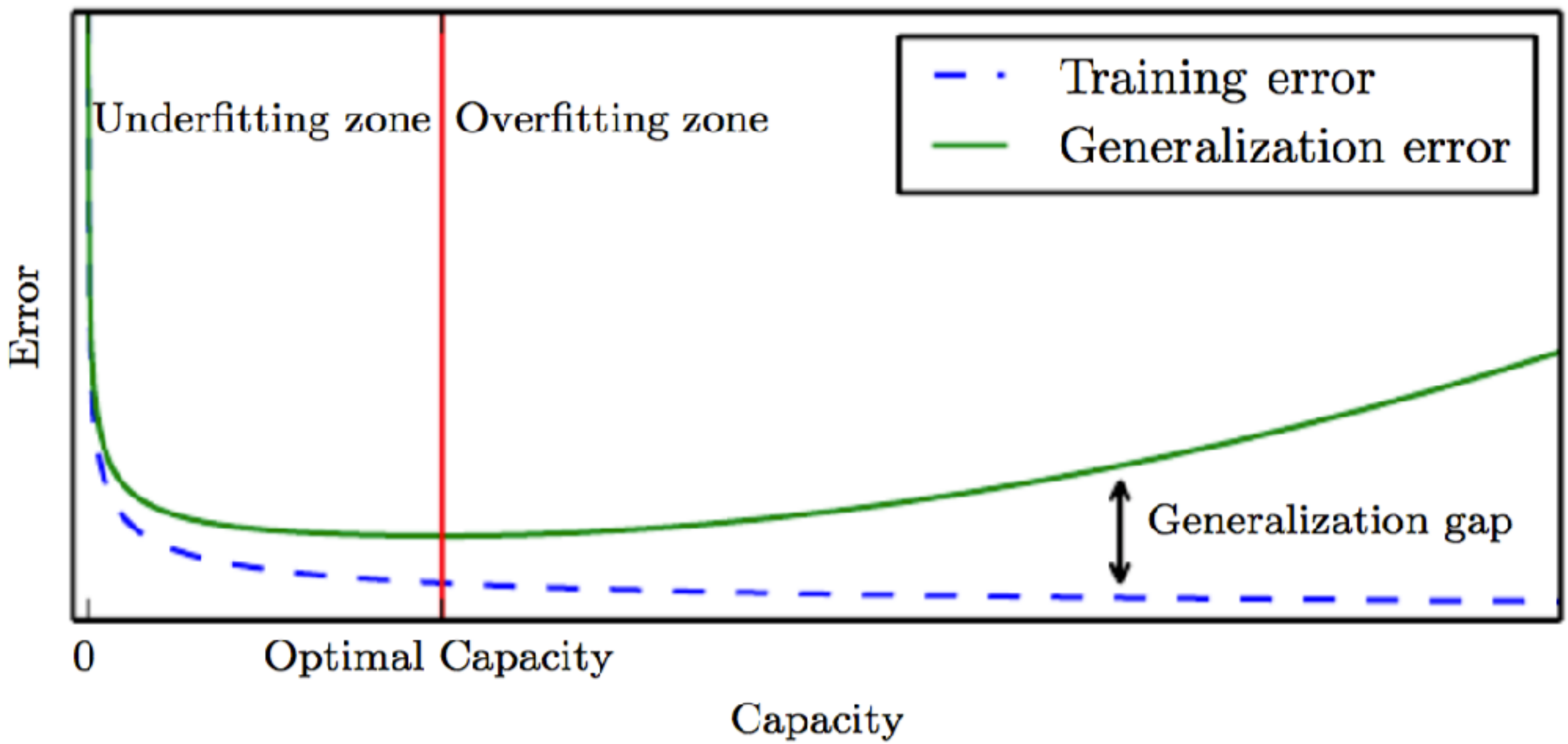
  - direction is a random variable, whose the expectation is the gradient to be estimated.
  - faster than batch gradient descent
- Minibatch SGD:
  - SGD on 10 to 100 examples (mini batch)
  - less noisy estimate of the gradient

Gradient-based approach



K. Duh, 2014

Over-fitting



Regularzation tricks to avoid overfitting

- Penalty terms In the training loss
- Data augmentation
- Dropout layers

Parameter norm penalization

- Regularized objective function:

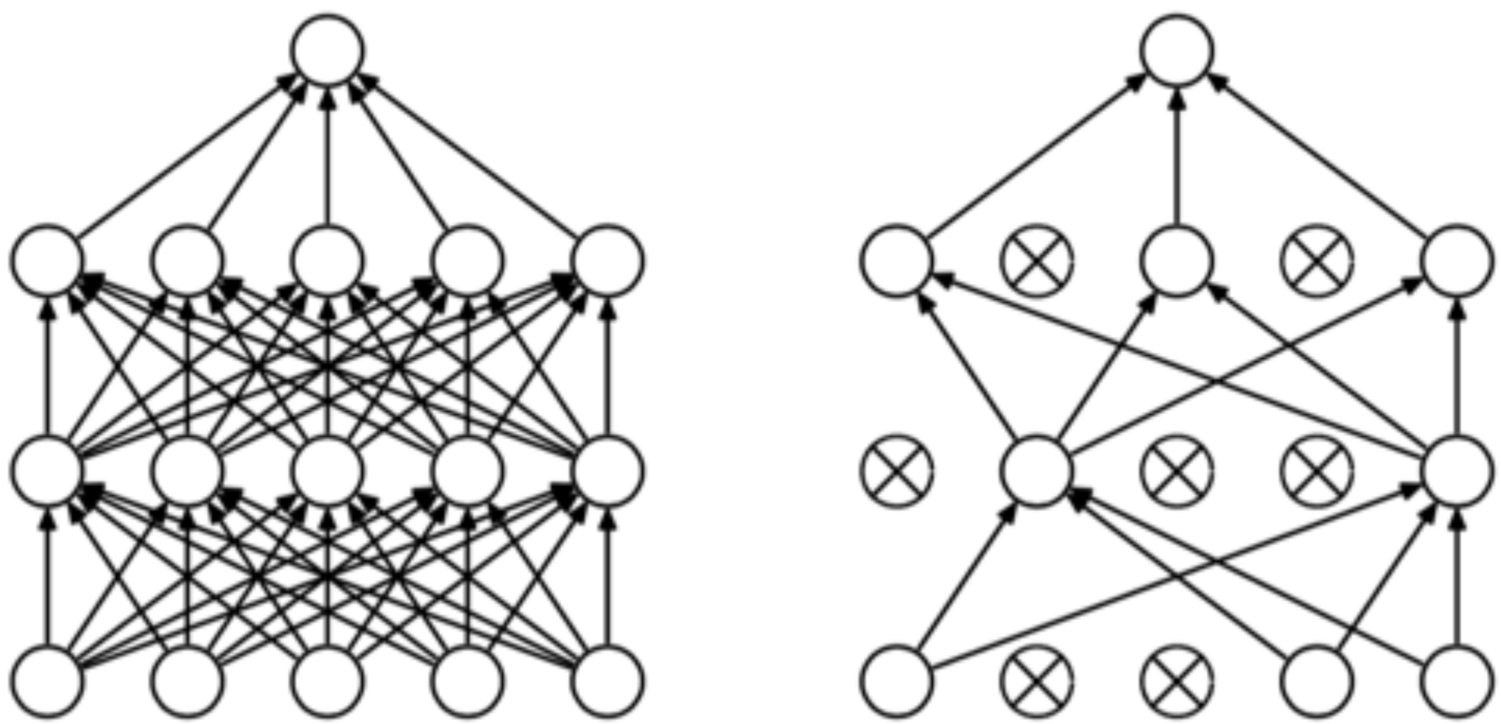
$$\tilde{J}(\theta) = J(\theta) + \alpha\Omega(\theta)$$

- L^2 norm: $\Omega(\theta) = \frac{1}{2}||w||_2^2$
- L^1 norm: $\Omega(\theta) = ||w||_1 = \sum_i |w_i|$

Data augmentation

- Purpose: improving model generalization error by training on more data
- Very efficient for object recognition
- How to:
  - apply (geometric) transformations on input data (such as translation, rotation, scaling for images).
  - noise injection

Dropout



(a) Standard Neural Net

(b) After applying dropout.

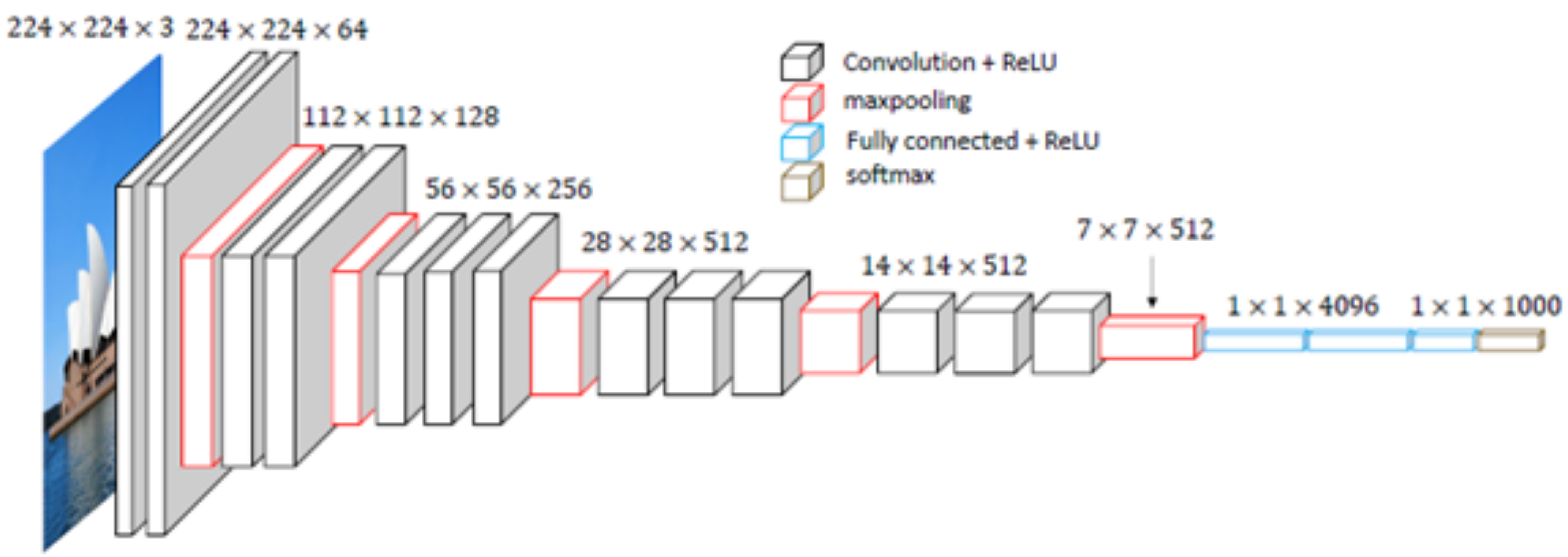
Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Srivastava et al., 2014

Convolutional Neural Networks

State-of-the-art NNs in computer vision

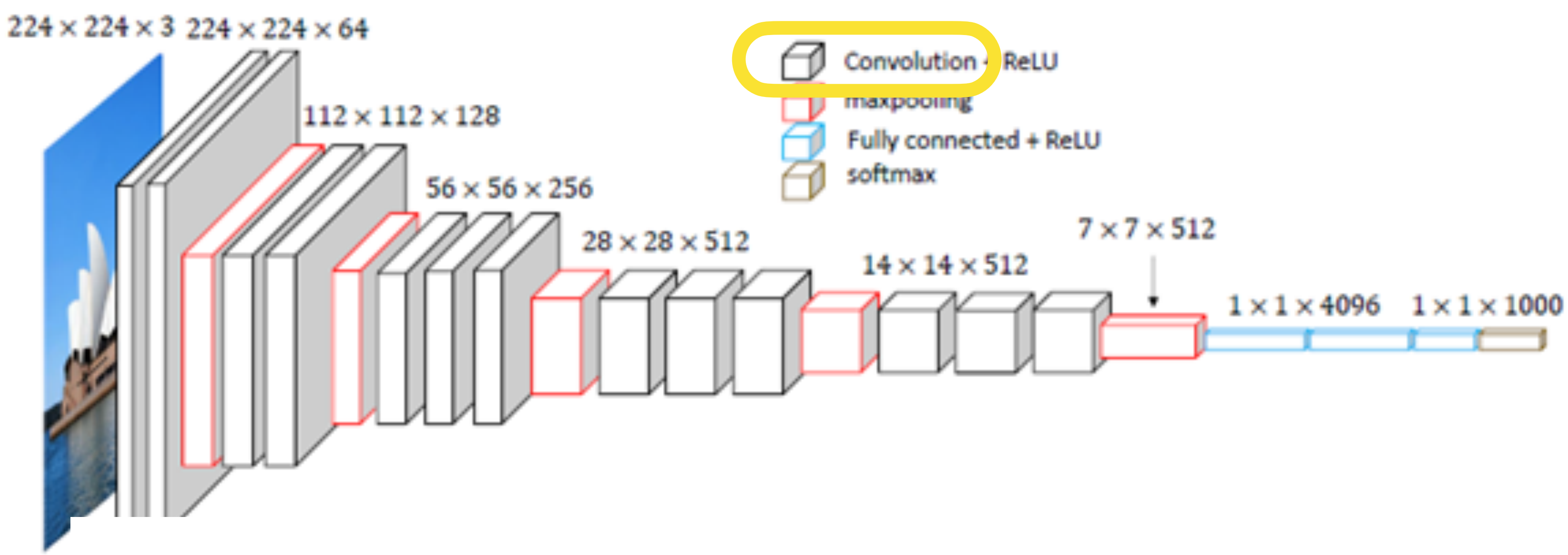
DL models are (in general) feedforward models. VGG16 as an illustration



Elementary components

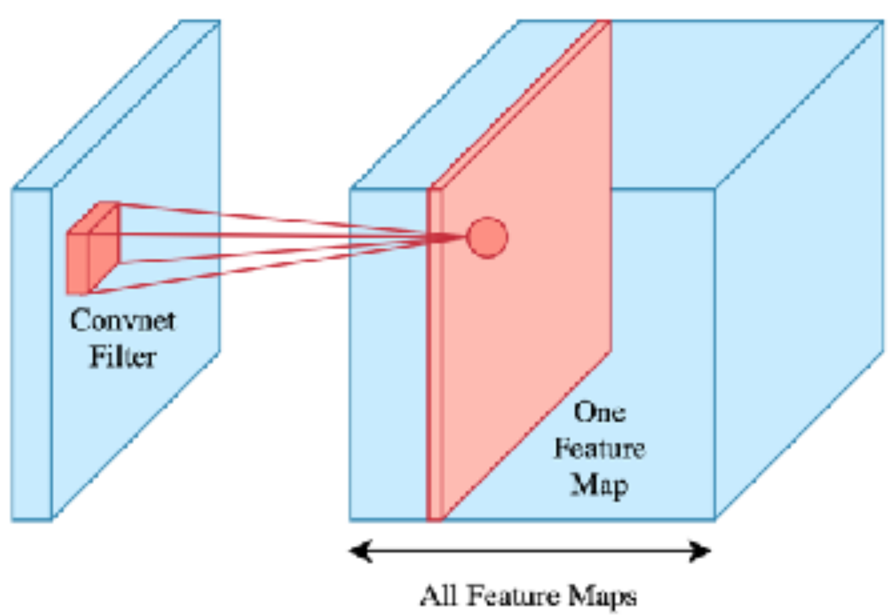
- Convolution layers
- Activation layers
- Pooling layers
- FC layers

Basics of DL models



Elementary components

Convolution layers	Activation layers	Pooling layers	Dense layers



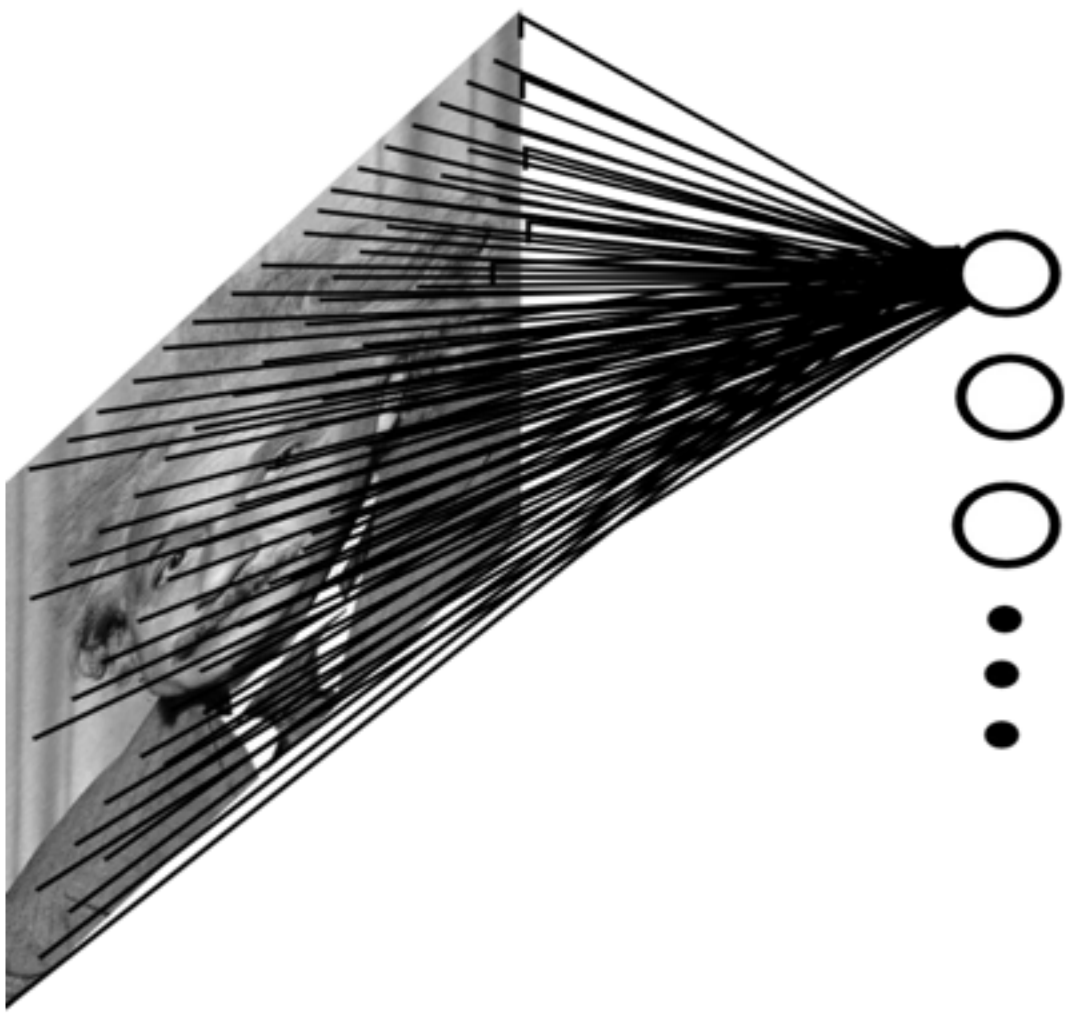

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

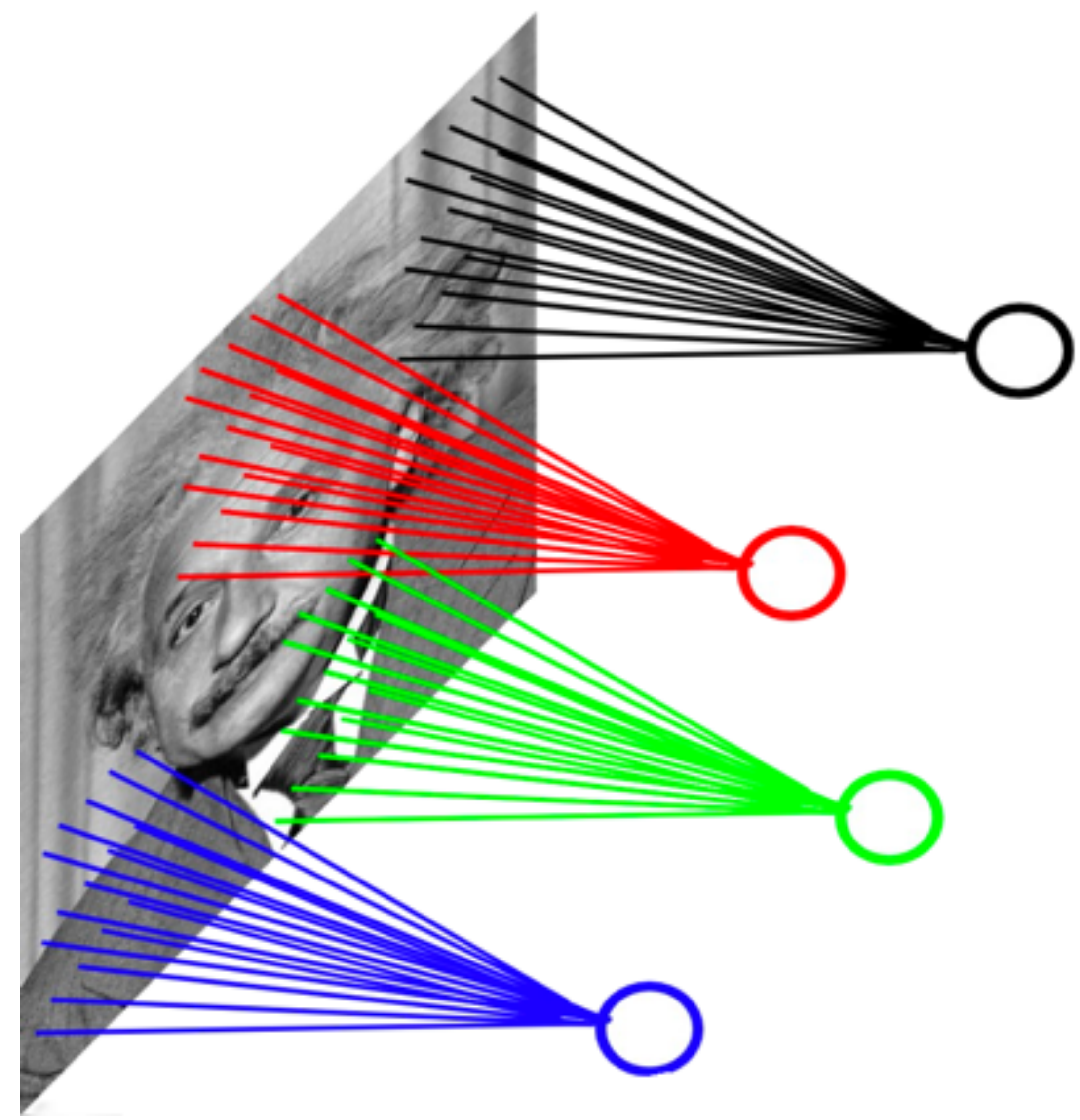
Number of parameters ?

Independent on the sizes of the input and output layer

Dense layer vs Conv layer

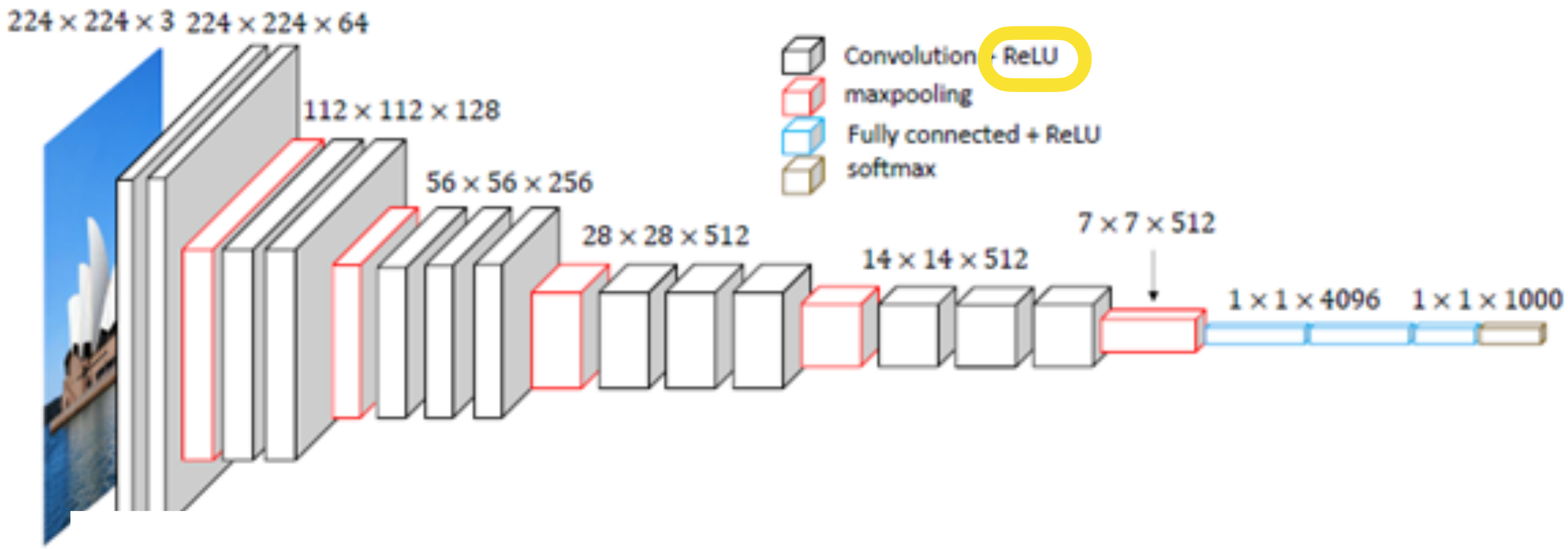


Dense layer



Convolutional layer

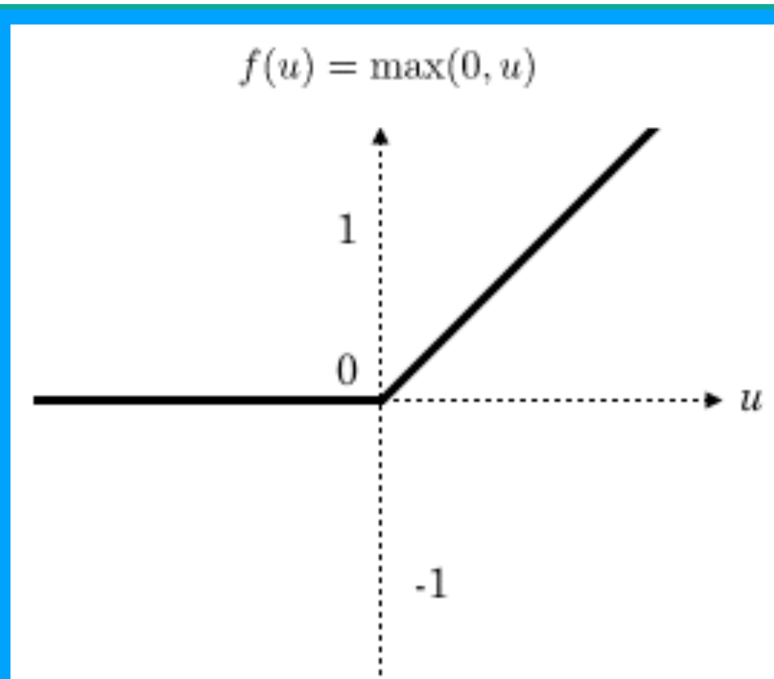
Basics of DL models

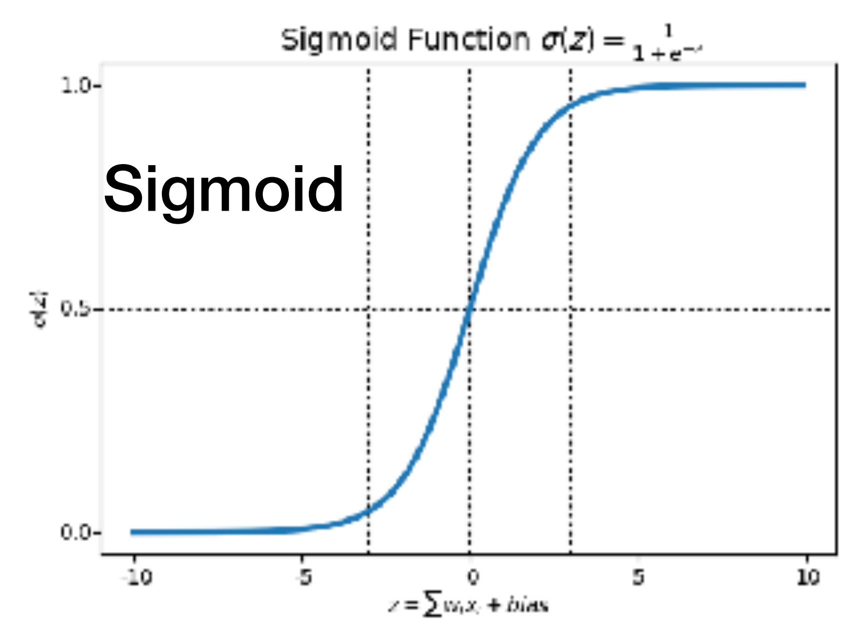


Elementary components

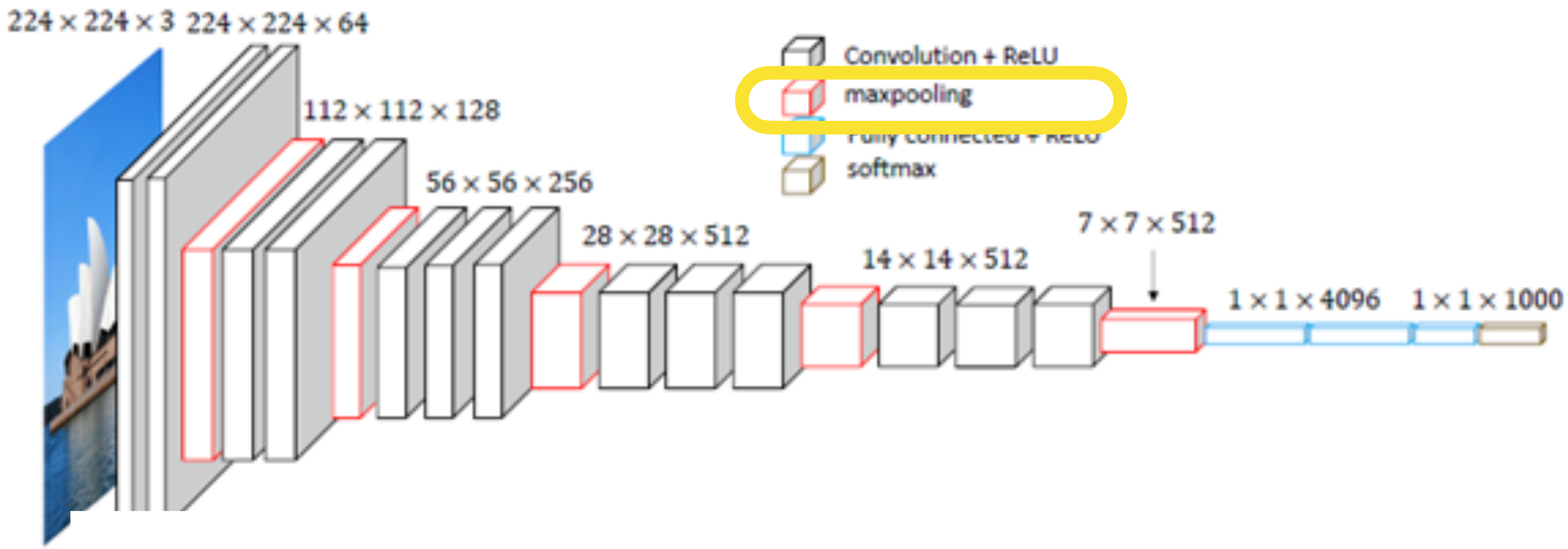
- Convolution layers
- Activation layers
- Pooling layers
- Dense layers

ReLU (Rectified Linear Unit)



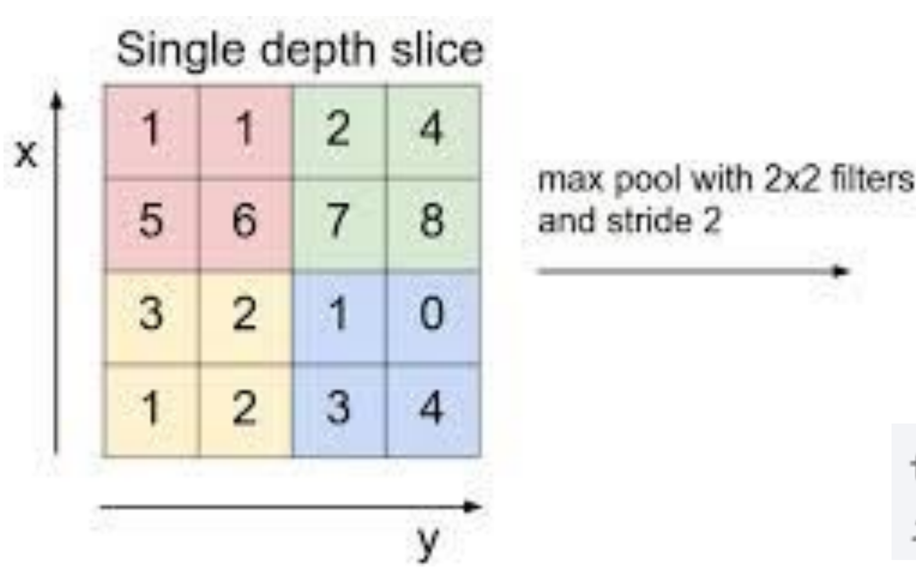


Basics of DL models

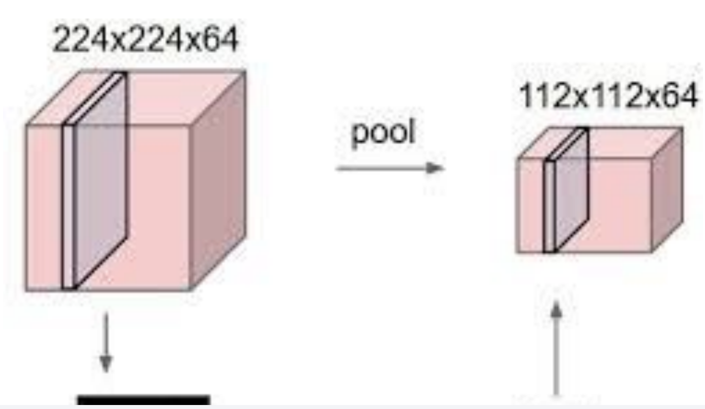


Elementary components: Convolution layers | Activation layers | Pooling layers | Dense layers

An example of max-pooling operator

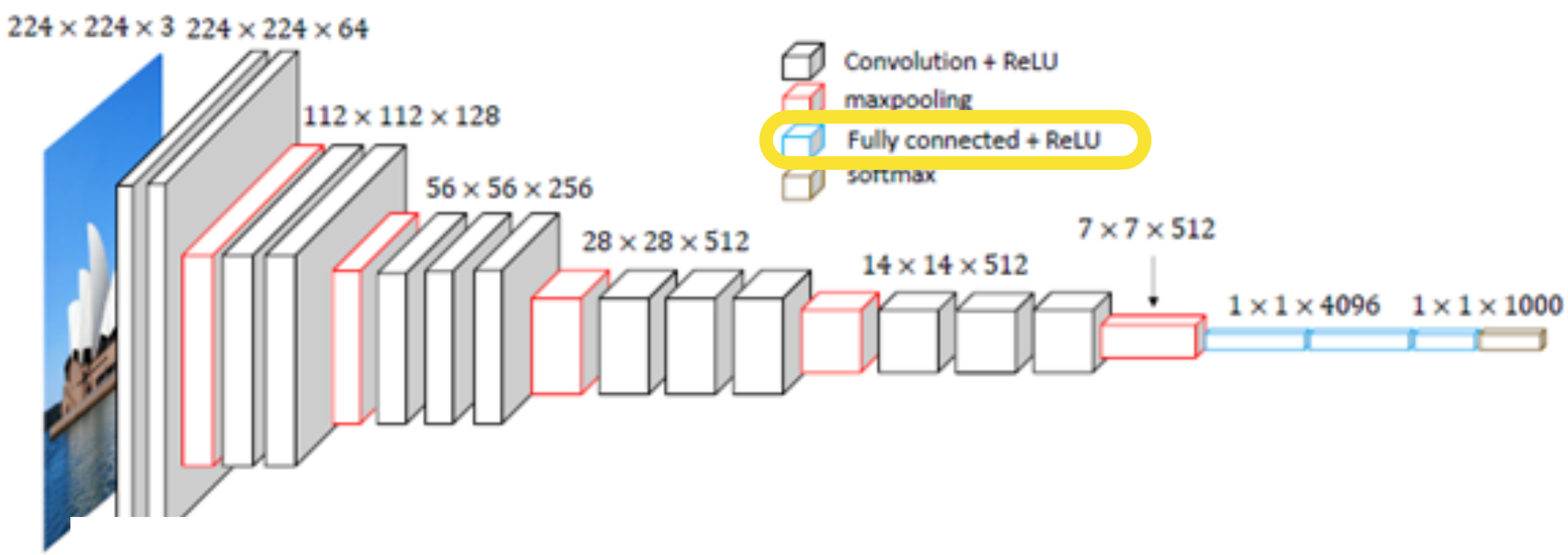


Pooling downsamples the input layer




```
torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1,
return_indices=False, ceil_mode=False)  [SOURCE]
```

Basics of DL models



Elementary components

- Convolution layers
- Activation layers
- Pooling layers
- Dense layers

Dense layers
or
Fully-connected (FC) layer

as in a classic MLP

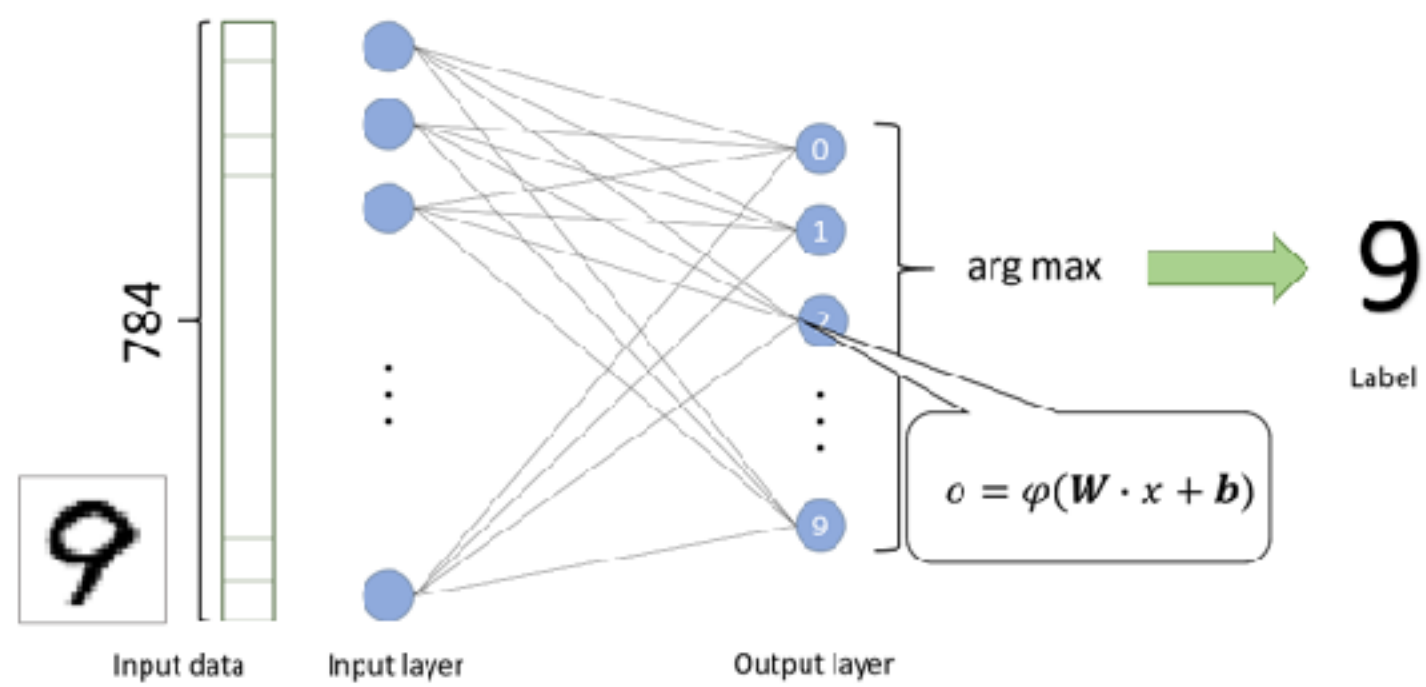
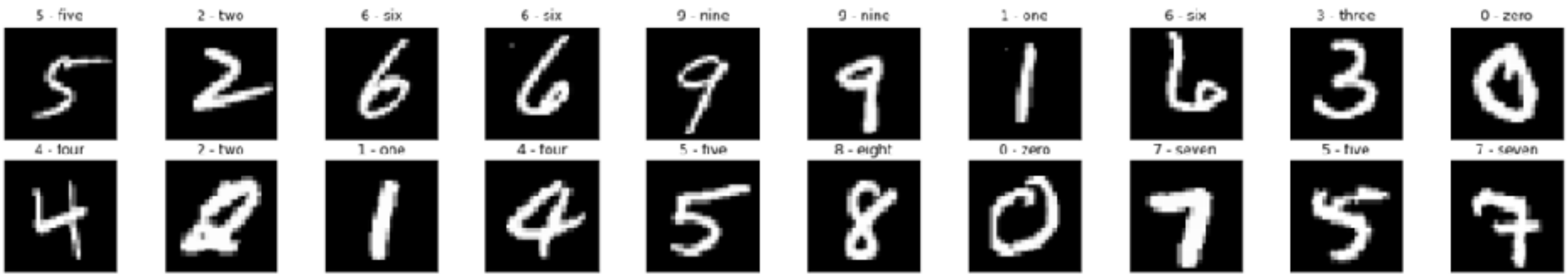
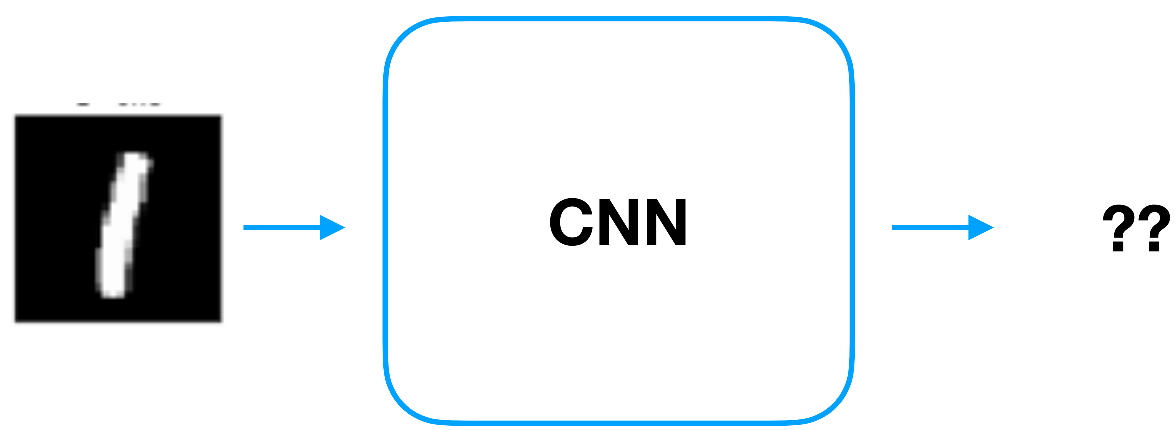


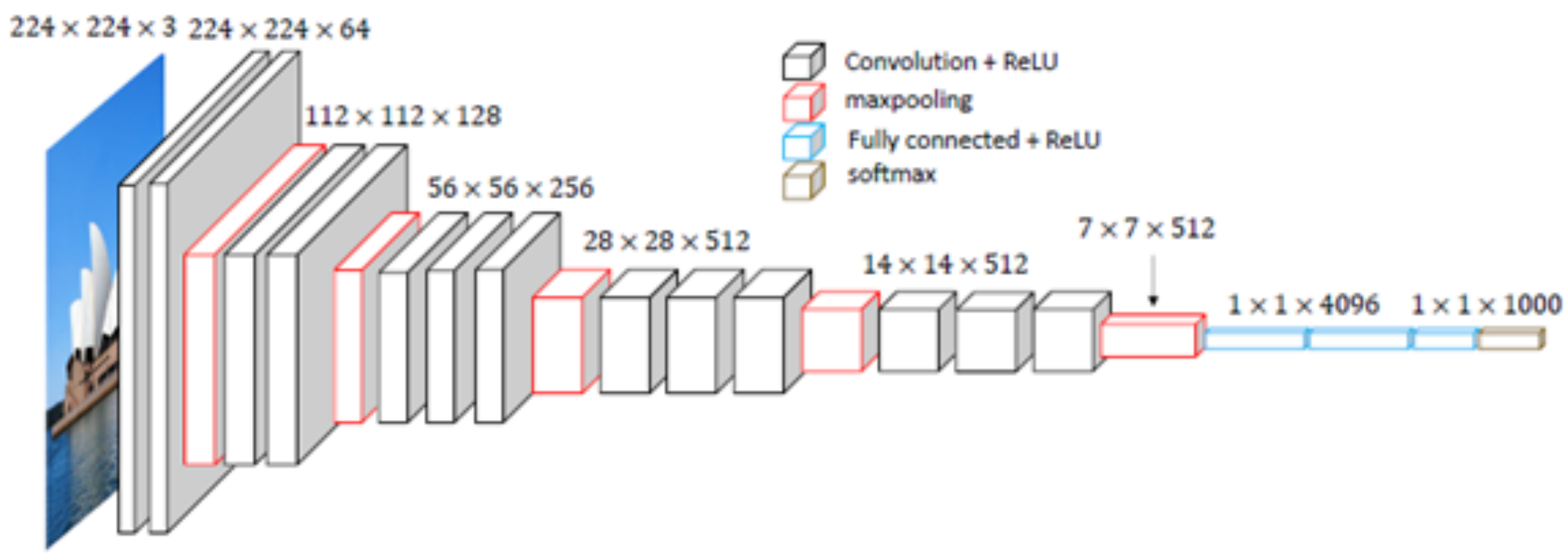
Image classification case-study with CNN





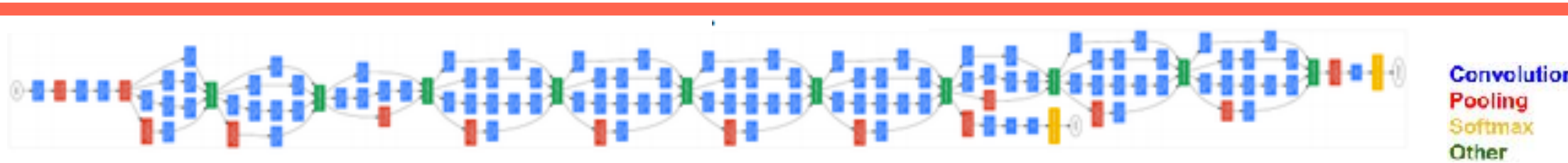
https://github.com/CIA-Oceanix/DLCourse_MOi_2022/blob/main/notebooks/notebook_MNIST_classification_MLP_CNN_students.ipynb

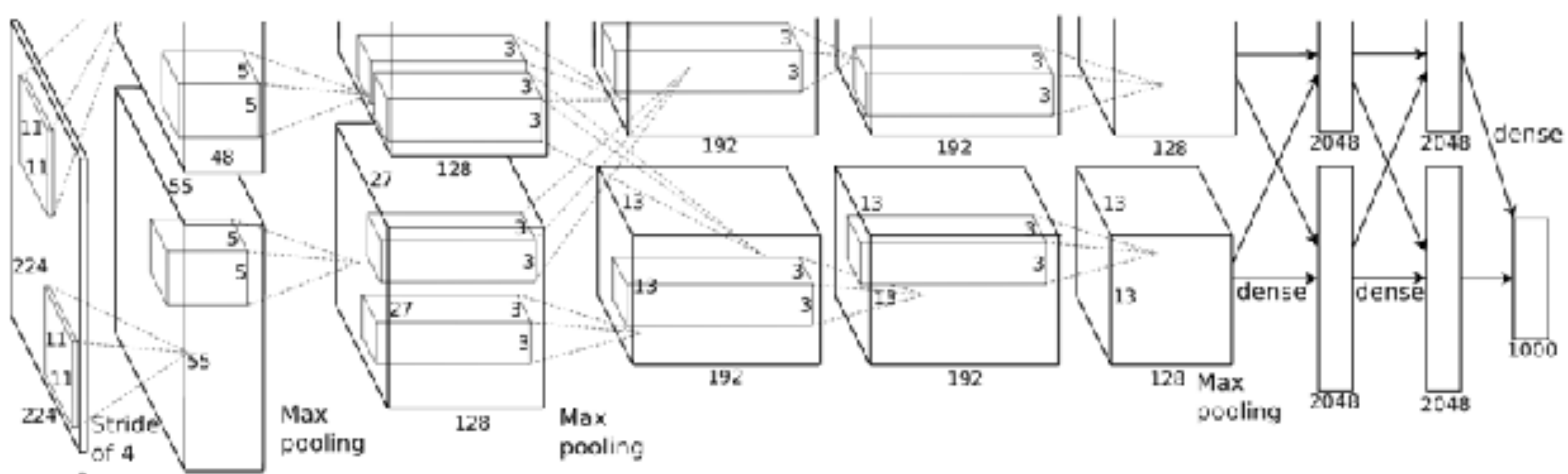
Examples of DL models for object recognition (2010-2020)



VGG16
(<100M of parameters)

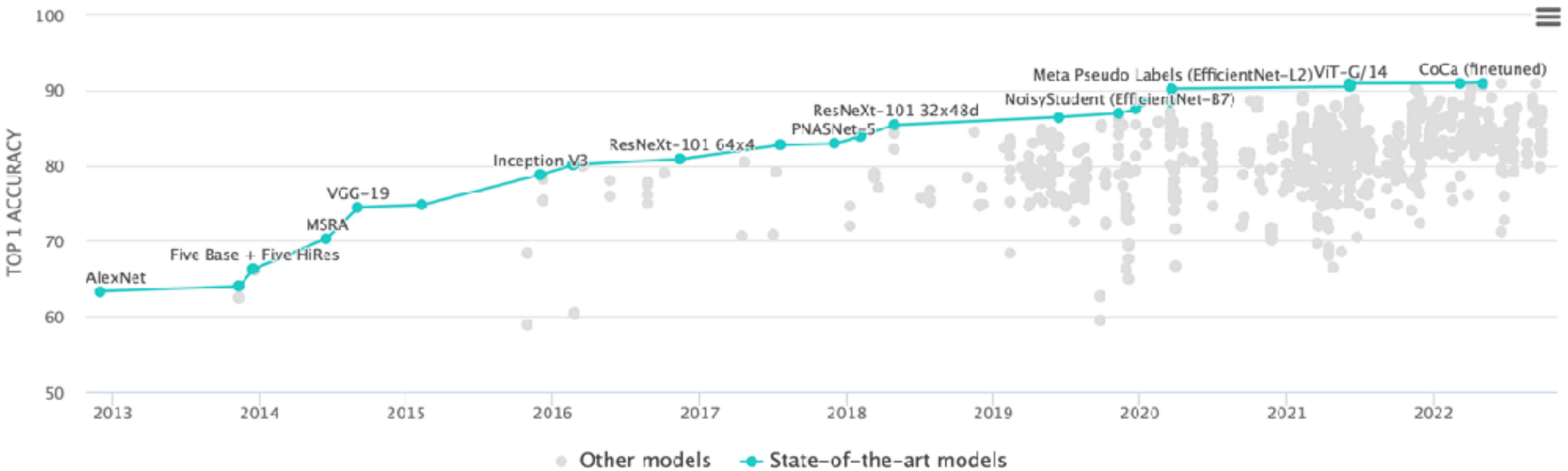
Google `
inception
(5M of parameters)





AlexNet
(60M of parameters)

DL and Benchmarking
(Data Challenges)

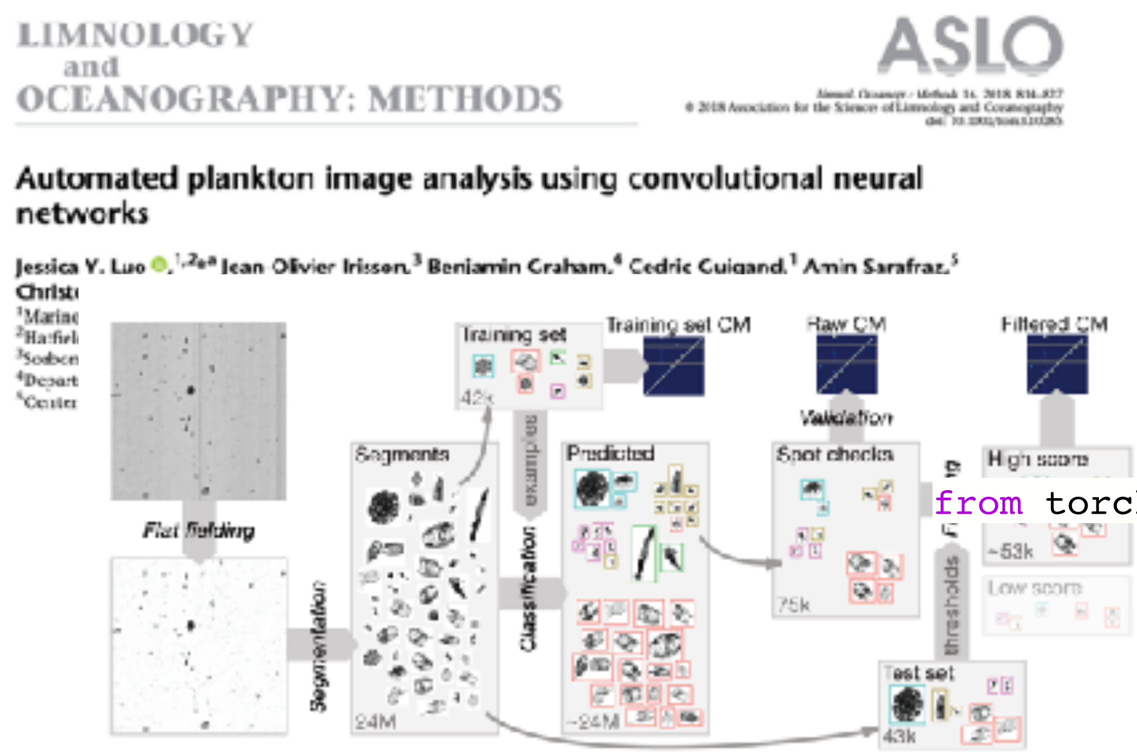


https://paperswithcode.com/sota/image-classification-on-imagenet

# of object classes: 1000
# of images > 1.2 M

Best accuracy score: ~91%
State-of-the-art architectures: CNN, Vision Transformers

CNN-based classification and Ocean Data

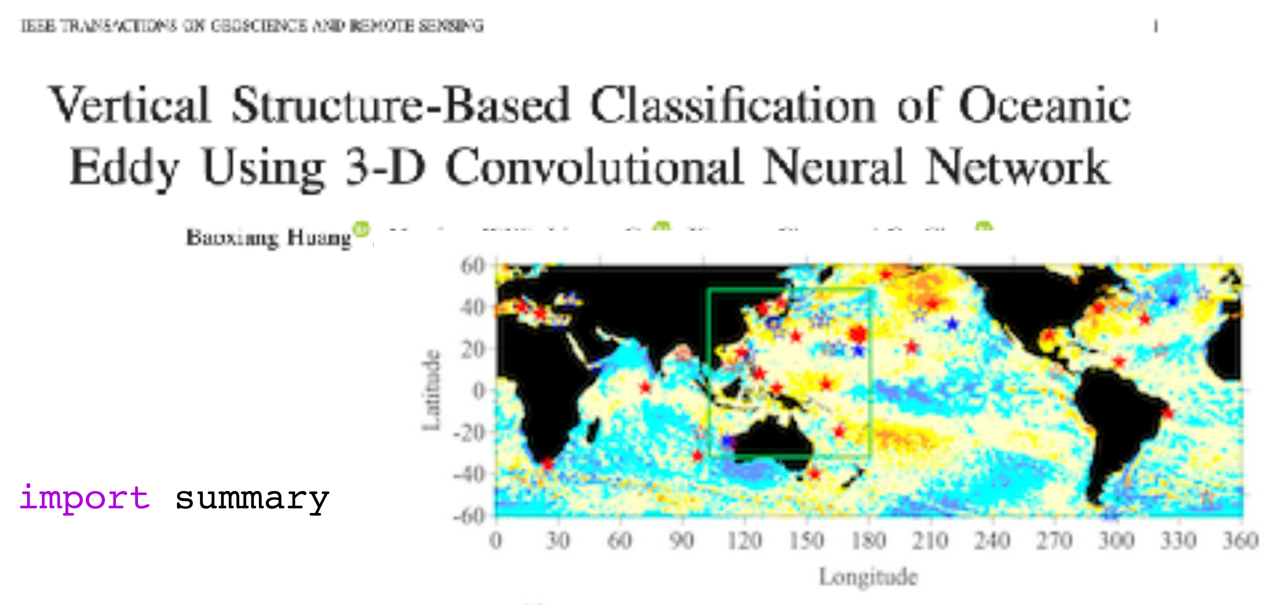

LIMNOLOGY and OCEANOGRAPHY: METHODS

ASLO

Automated plankton image analysis using convolutional neural networks

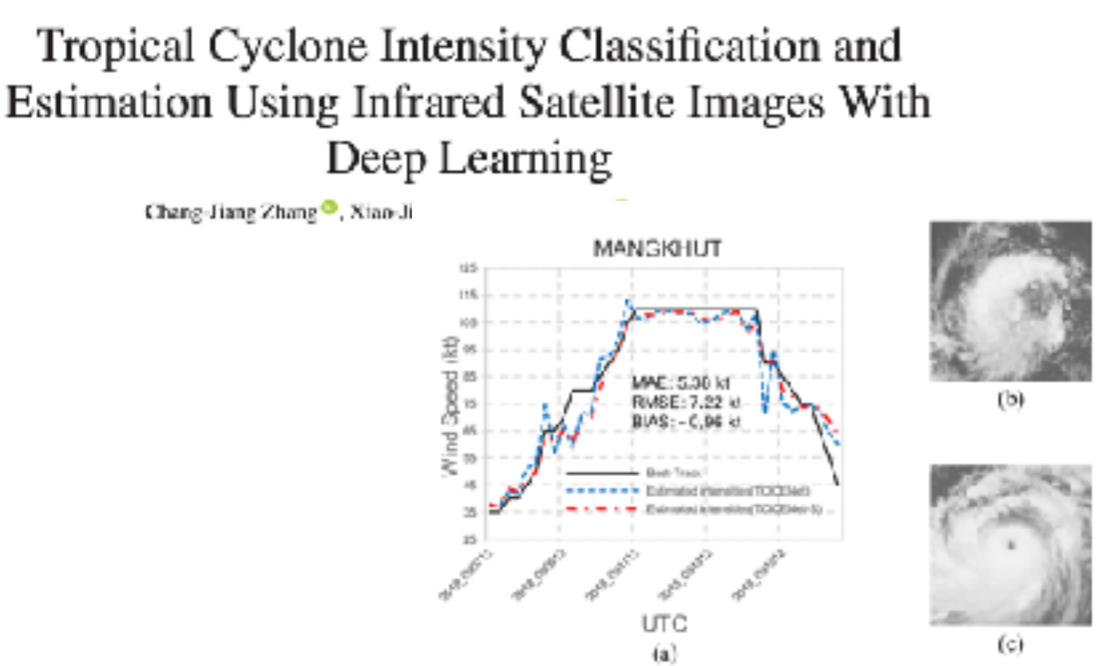
Jessica Y. Luo, Jean Olivier Irisson, Benjamin Graham, Cedric Guigand, Amin Sarafraz


```
from torchsummary import summary
```

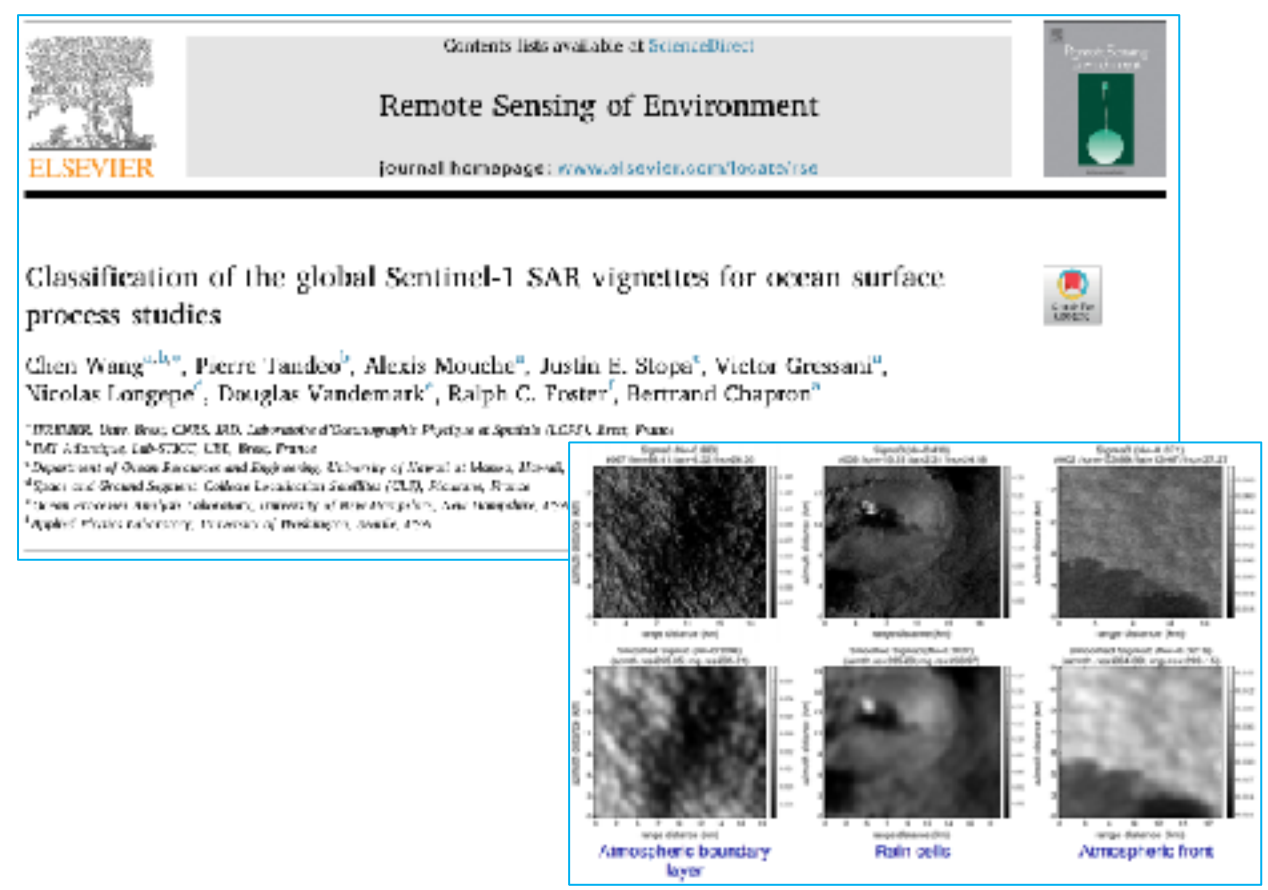

IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING

Vertical Structure-Based Classification of Oceanic Eddy Using 3-D Convolutional Neural Network

Baoxiang Huang


Tropical Cyclone Intensity Classification and Estimation Using Infrared Satellite Images With Deep Learning

Chang-Jiang Zhang, Xian-Ji


Contents lists available at ScienceDirect

Remote Sensing of Environment

journal homepage: www.elsevier.com/locate/rse

ELSEVIER

Classification of the global Sentinel-1 SAR vignettes for ocean surface process studies

Chen Wang, Pierre Tandeo, Alexis Mouche, Justin E. Stopa, Victor Gressani, Nicolas Longepe, Douglas Vandemark, Ralph C. Foster, Bertrand Chapron

Fine-tuning from pre-trained models

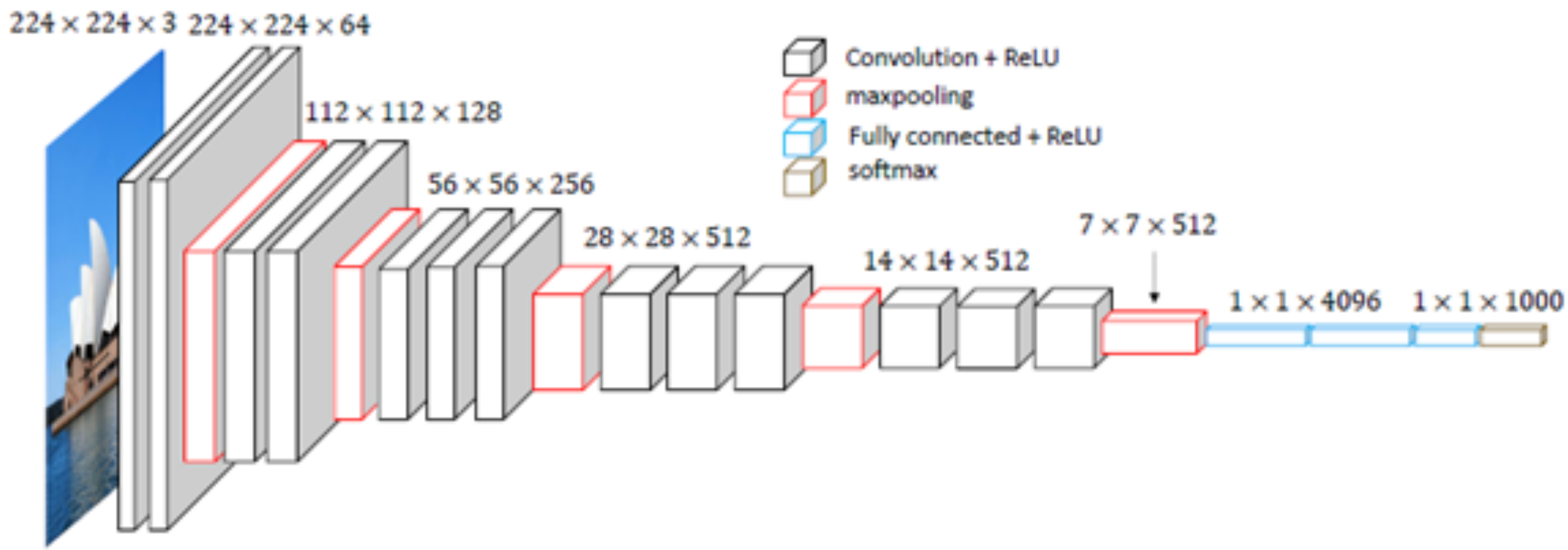


Figure 2: The architecture of VGG16 model .

General idea: the first layers involve generic feature extraction step and the last block can be regarded as a dataset-specific classification block.

Fine-tuning from pre-trained models

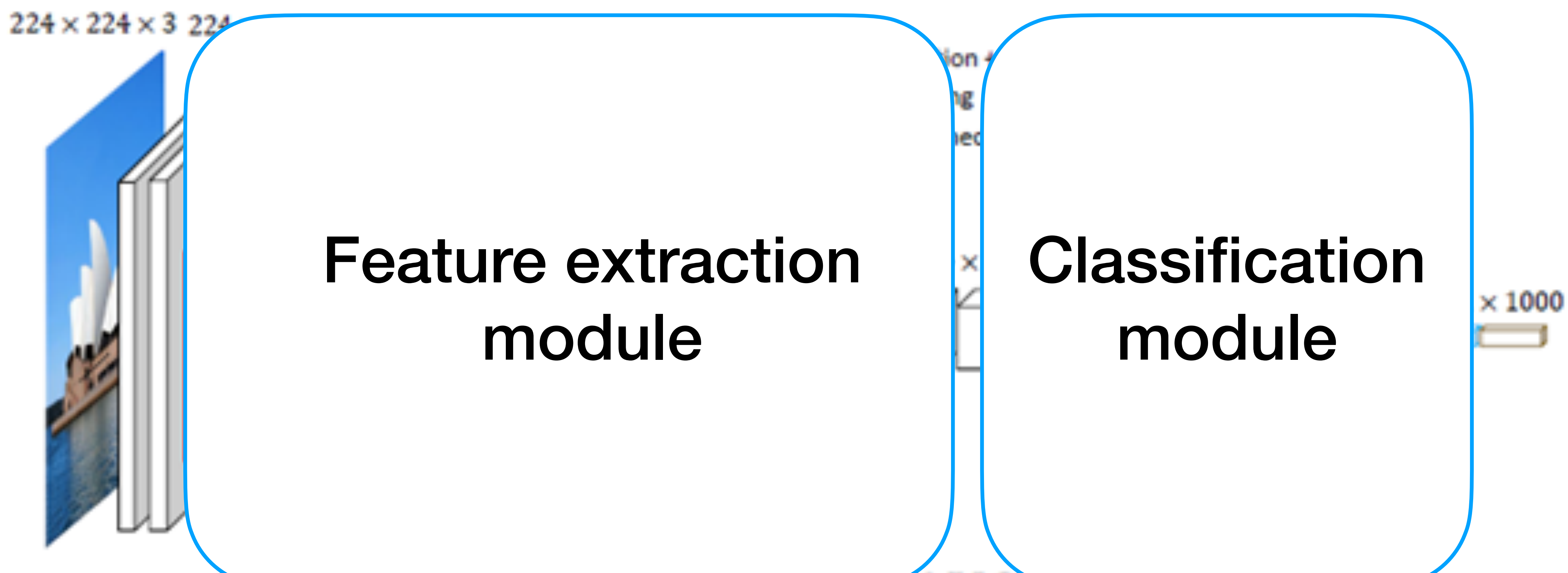
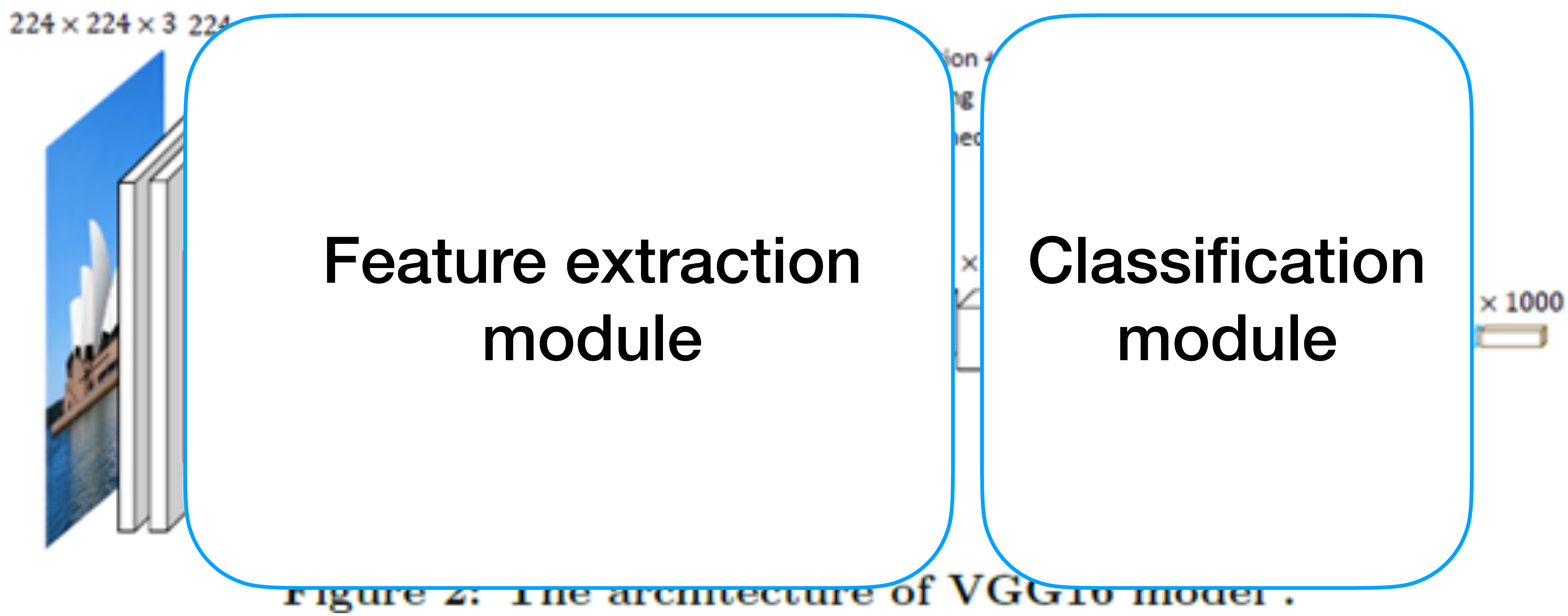


Figure 2: The architecture of VGG16 model .

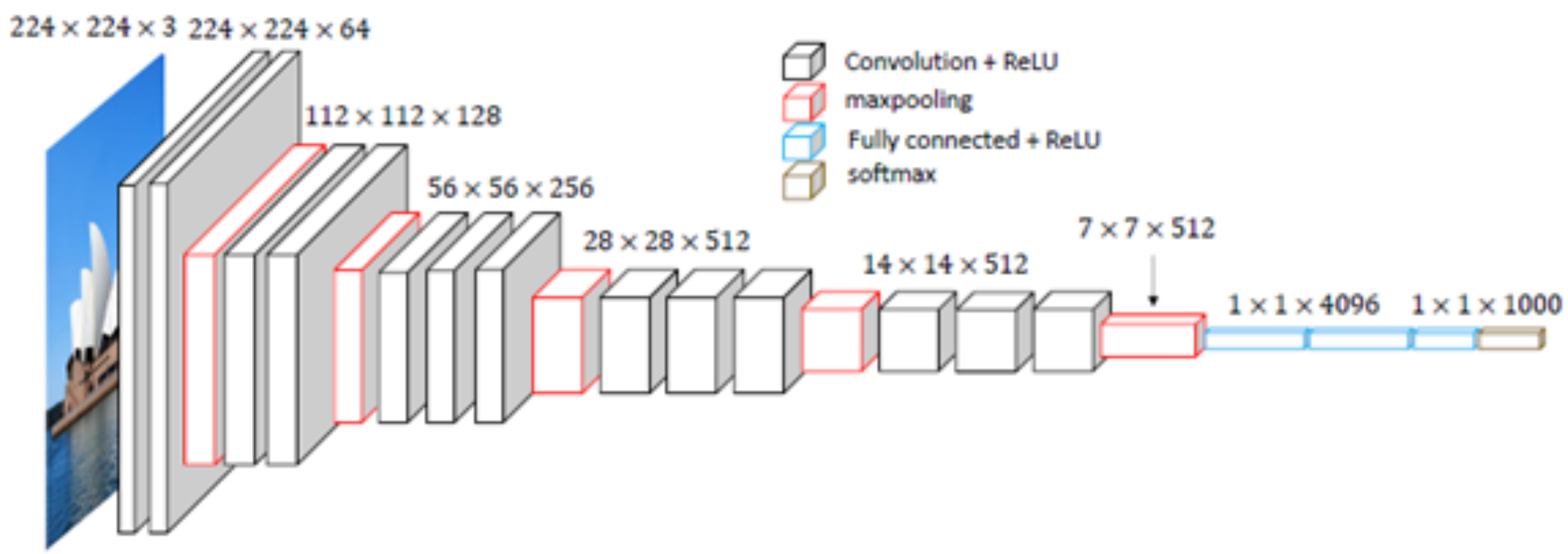
General idea: the first layers involve generic feature extraction step and the last block can be regarded as a dataset-specific classification block.

Fine-tuning from pre-trained models



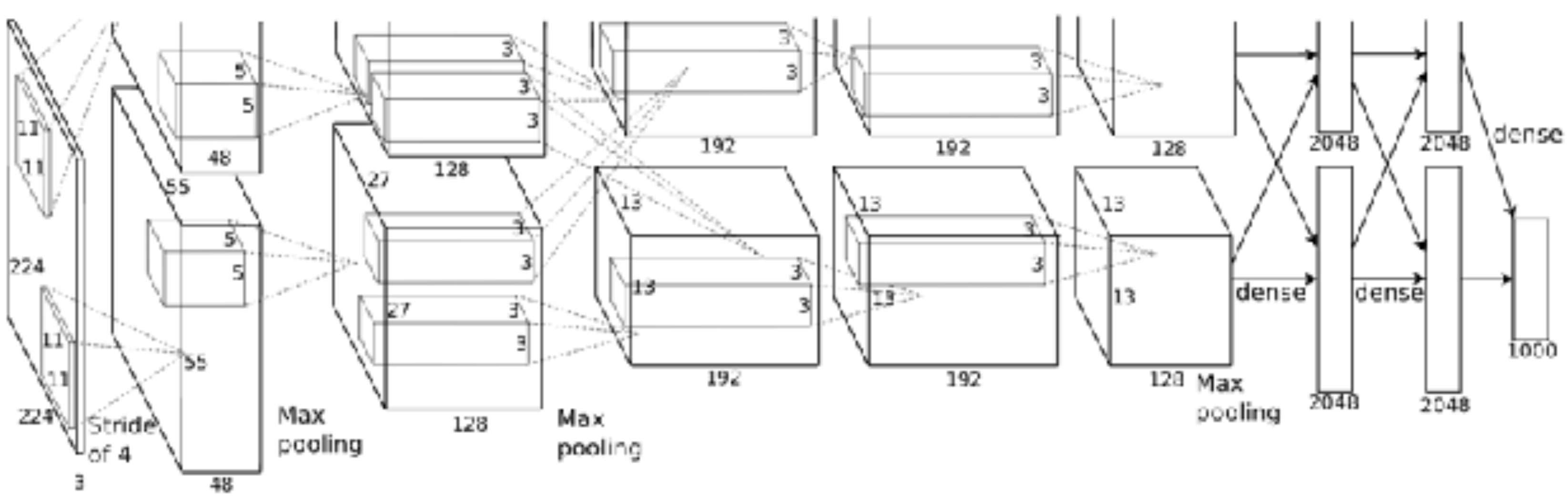
https://github.com/CIA-Oceanix/DLCourse_MOi_2022/blob/main/notebooks/notebook_MNIST_classification_MLP_CNN_TransferLearning_students.ipynb

Examples of DL models for object recognition (2010-2020)



VGG16
(<100M of parameters)

Google `
inception
(5M of parameters)



AlexNet
(60M of parameters)

Lecture. #2
Things to know (CNN)

- Convolution layers
- Pooling layers
- Activation layers
- Dropout layers
- Padding and stride
- Fully-Connected/Dense layers
- Fine-tuning
- Over-fitting
- Data augmentation